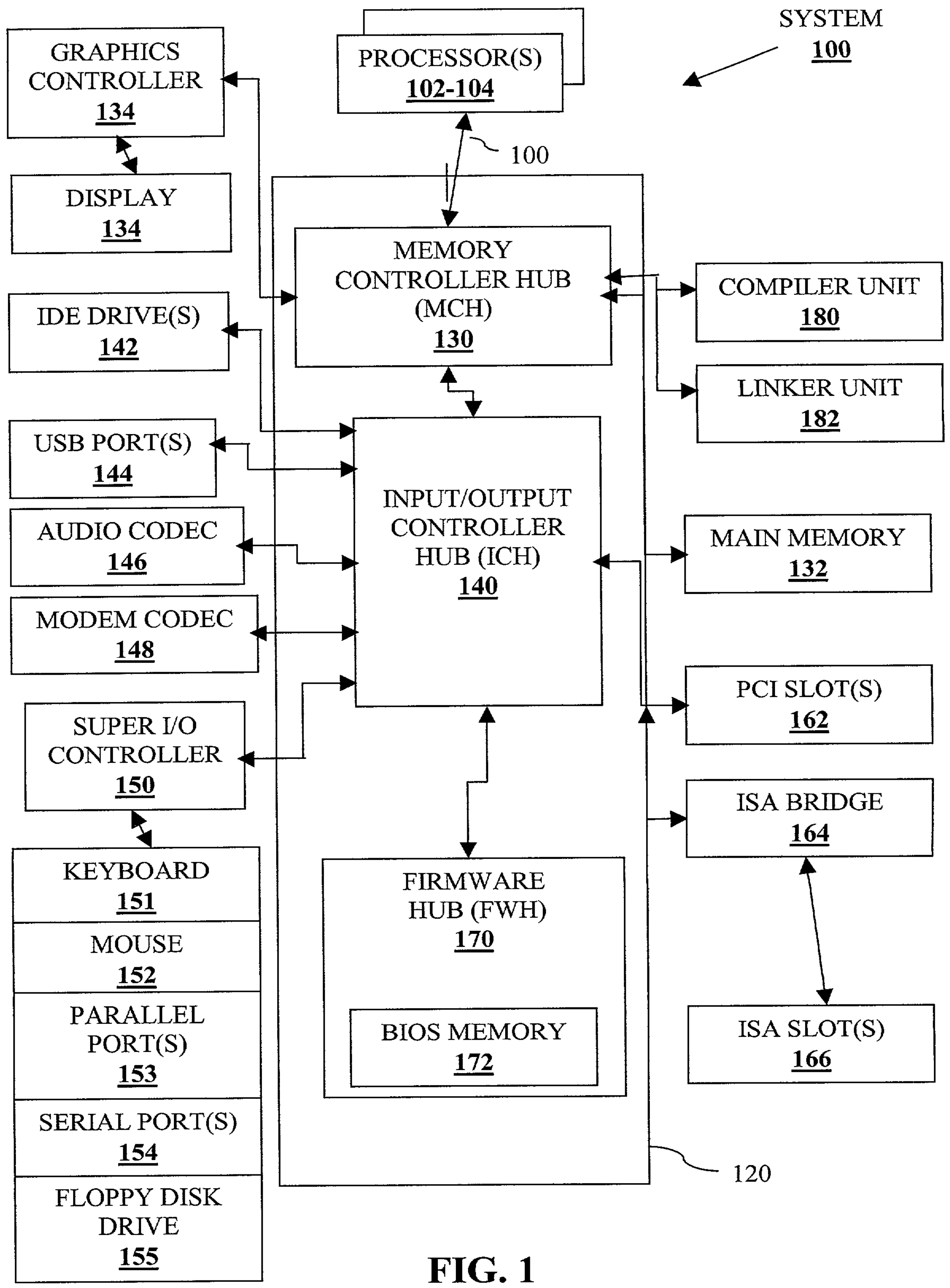
SYSTEM 100
PROCESSOR(S) 102-104
100
GRAPHICS CONTROLLER 134
DISPLAY 134
IDE DRIVE(S) 142
USB PORT(S) 144
AUDIO CODEC 146
MODEM CODEC 148
SUPER I/O CONTROLLER 150
KEYBOARD 151
MOUSE 152
PARALLEL PORT(S) 153
SERIAL PORT(S) 154
FLOPPY DISK DRIVE 155
MEMORY CONTROLLER HUB (MCH) 130
INPUT/OUTPUT CONTROLLER HUB (ICH) 140
FIRMWARE HUB (FWH) 170
BIOS MEMORY 172
COMPILER UNIT 180
LINKER UNIT 182
MAIN MEMORY 132
PCI SLOT(S) 162
ISA BRIDGE 164
ISA SLOT(S) 166
120

FIG. 1

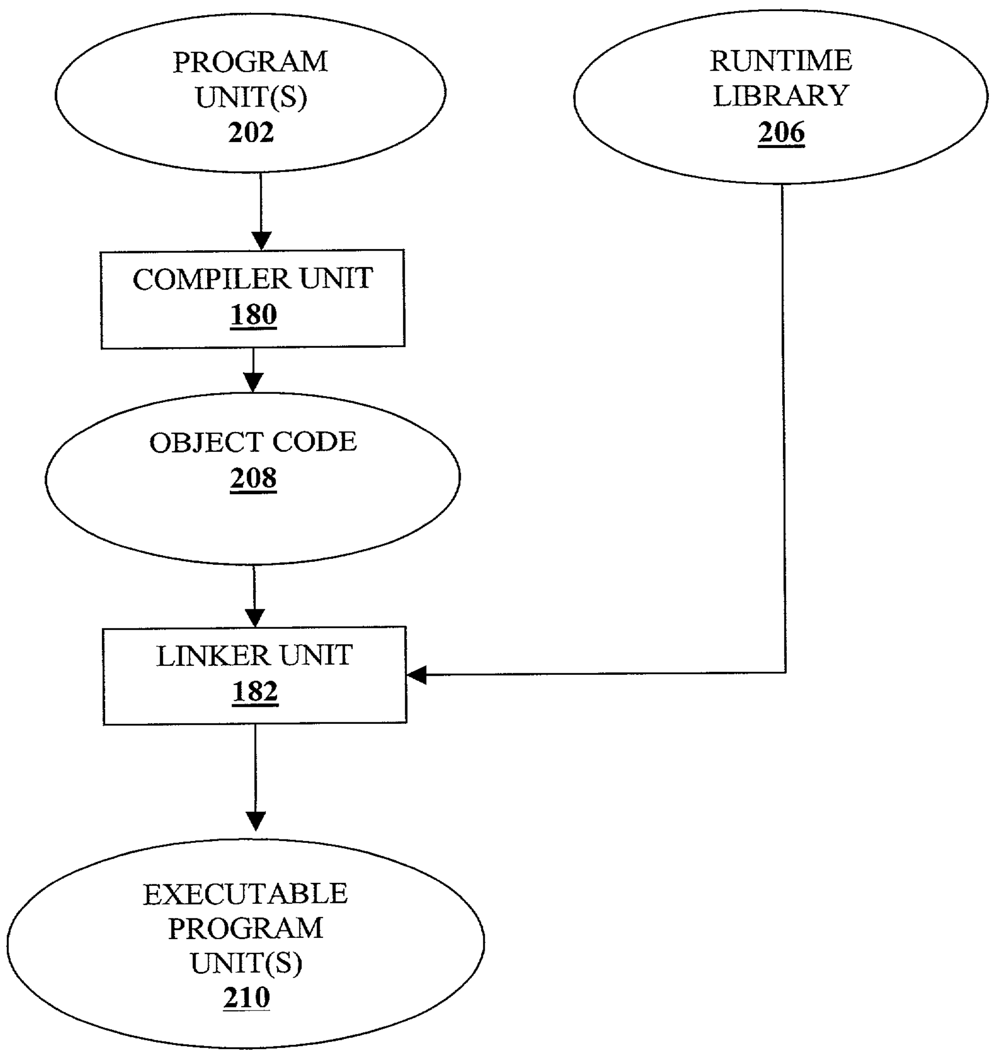
PROGRAM UNIT(S) 202
RUNTIME LIBRARY 206
COMPILER UNIT 180
OBJECT CODE 208
LINKER UNIT 182
EXECUTABLE PROGRAM UNIT(S) 210

FIG. 2

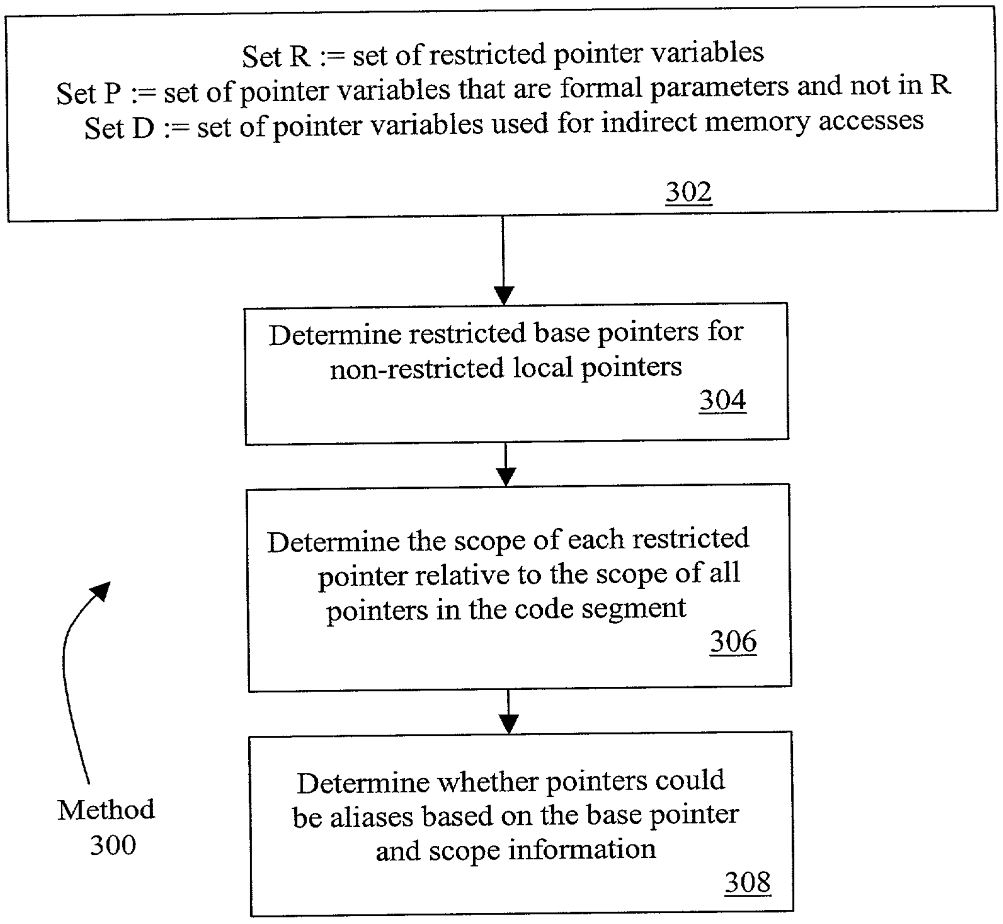
Set R := set of restricted pointer variables
Set P := set of pointer variables that are formal parameters and not in R
Set D := set of pointer variables used for indirect memory accesses
302
Determine restricted base pointers for non-restricted local pointers
304
Determine the scope of each restricted pointer relative to the scope of all pointers in the code segment
306
Determine whether pointers could be aliases based on the base pointer and scope information
308
Method 300

FIG. 3

Program 400

```
410 void bar( float * restrict a, float * x, int i, int j, int k ) {
415   a[0] = x[0];
420   {
425     float * restrict b = a-k;
430     float * restrict c = x+k;
435     float * y = b+i;
440     c[i] = *y;
445   }
450   {
455     float * restrict d = a;
460     {
470       float * restrict e = x;
475       d[j] = e[j];
480     }
490   }
495 }
```

FIG. 4

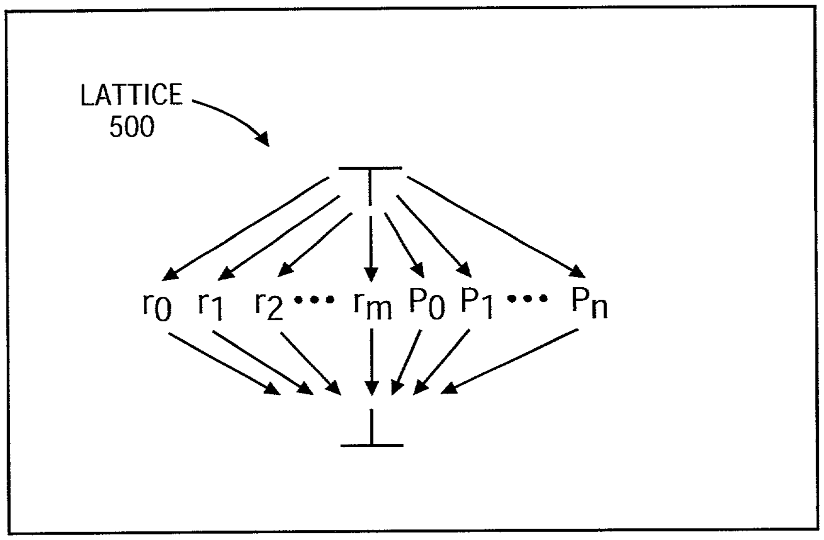

FIG. 5

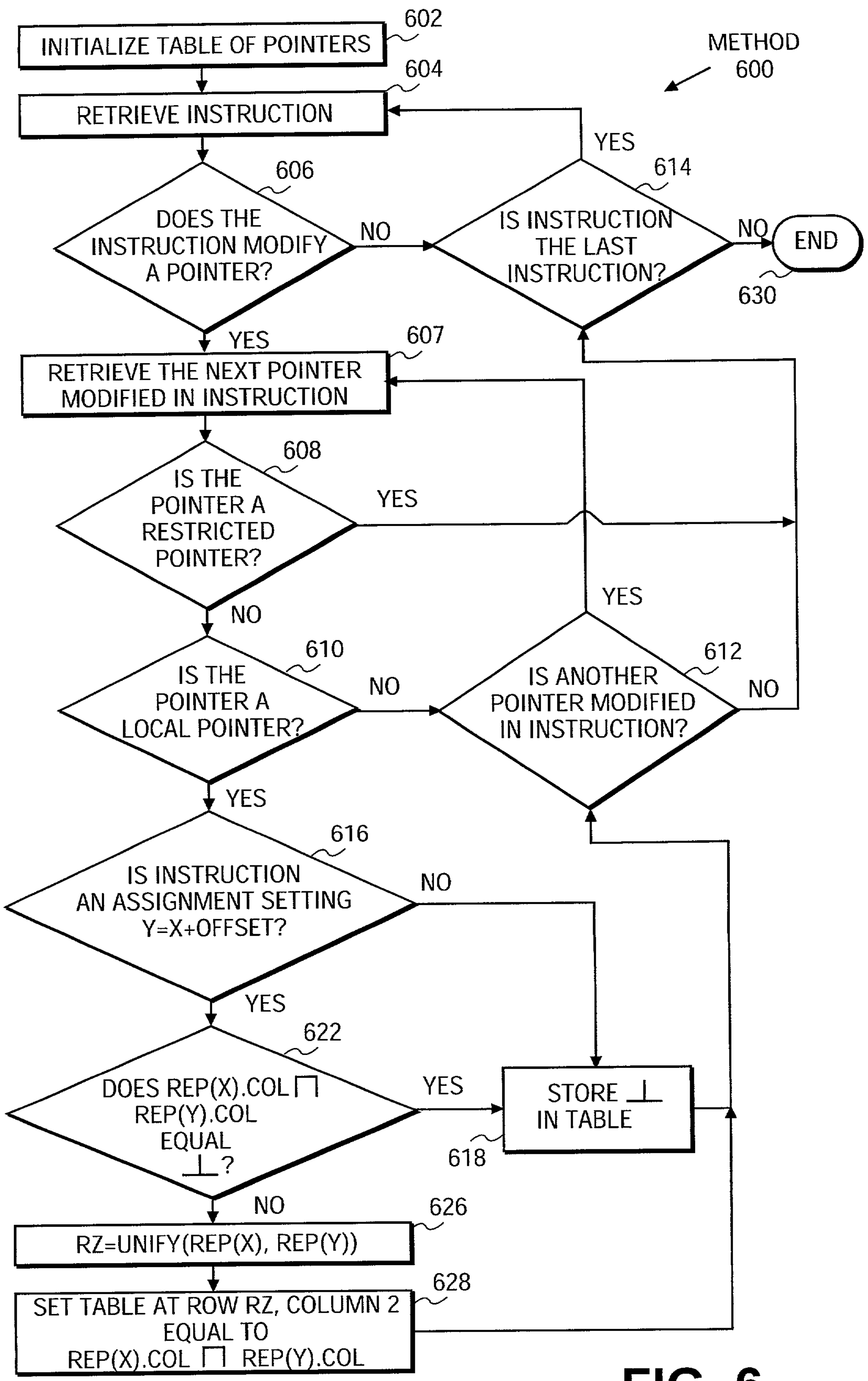
METHOD 600
INITIALIZE TABLE OF POINTERS
602
RETRIEVE INSTRUCTION
604
DOES THE INSTRUCTION MODIFY A POINTER?
606
NO
YES
IS INSTRUCTION THE LAST INSTRUCTION?
614
YES
NO
END
630
RETRIEVE THE NEXT POINTER MODIFIED IN INSTRUCTION
607
IS THE POINTER A RESTRICTED POINTER?
608
YES
NO
IS THE POINTER A LOCAL POINTER?
610
NO
YES
IS ANOTHER POINTER MODIFIED IN INSTRUCTION?
612
YES
NO
IS INSTRUCTION AN ASSIGNMENT SETTING Y=X+OFFSET?
616
NO
YES
DOES REP(X).COL ⊓ REP(Y).COL EQUAL ⊥?
622
YES
NO
STORE ⊥ IN TABLE
618
RZ=UNIFY(REP(X), REP(Y))
626
SET TABLE AT ROW RZ, COLUMN 2 EQUAL TO REP(X).COL ⊓ REP(Y).COL
628

FIG. 6

```
procedure FLOW_WALK

702  for each pointer variable w do
704    if $w \in (R \cup P)$ then
706      REP(w).col = w;
708    else
710      REP(w).col = $\top$;
712  enddo                                                     702

714  for each instruction do
716    for each pointer variable y that might be modified by the instruction
718      if y is pointer variable that is restrict qualified then
720        //Ignore it.
722      else
724        if y is a local pointer variable then
726          if instruction is assignment that sets y to adjustment of x then
728            if REP(x).col $\sqcap$ REP(y).col = $\perp$ then
730              // Do not unify.  Doing so just loses information.
732              REP(y).col = $\perp$
734            else
736              //Target of y is same as target of x
738              rz = UNIFY(REP(y), REP(x) );
740              rz.col = REP(x).col $\sqcap$ REP(y).col;
742            endif
744          else
746            //Target of y is unknown
748            REP(y).col := $\perp$;
750          endif
752        endif
756    enddo
758  enddo

760 end FLOW_WALK                                                704
```

Pseudo Code 700

FIG. 7

Table 800

Pointer	REP(...).col
a	a
b	b
c	c
d	d
e	e
x	x
y	b

FIG. 8

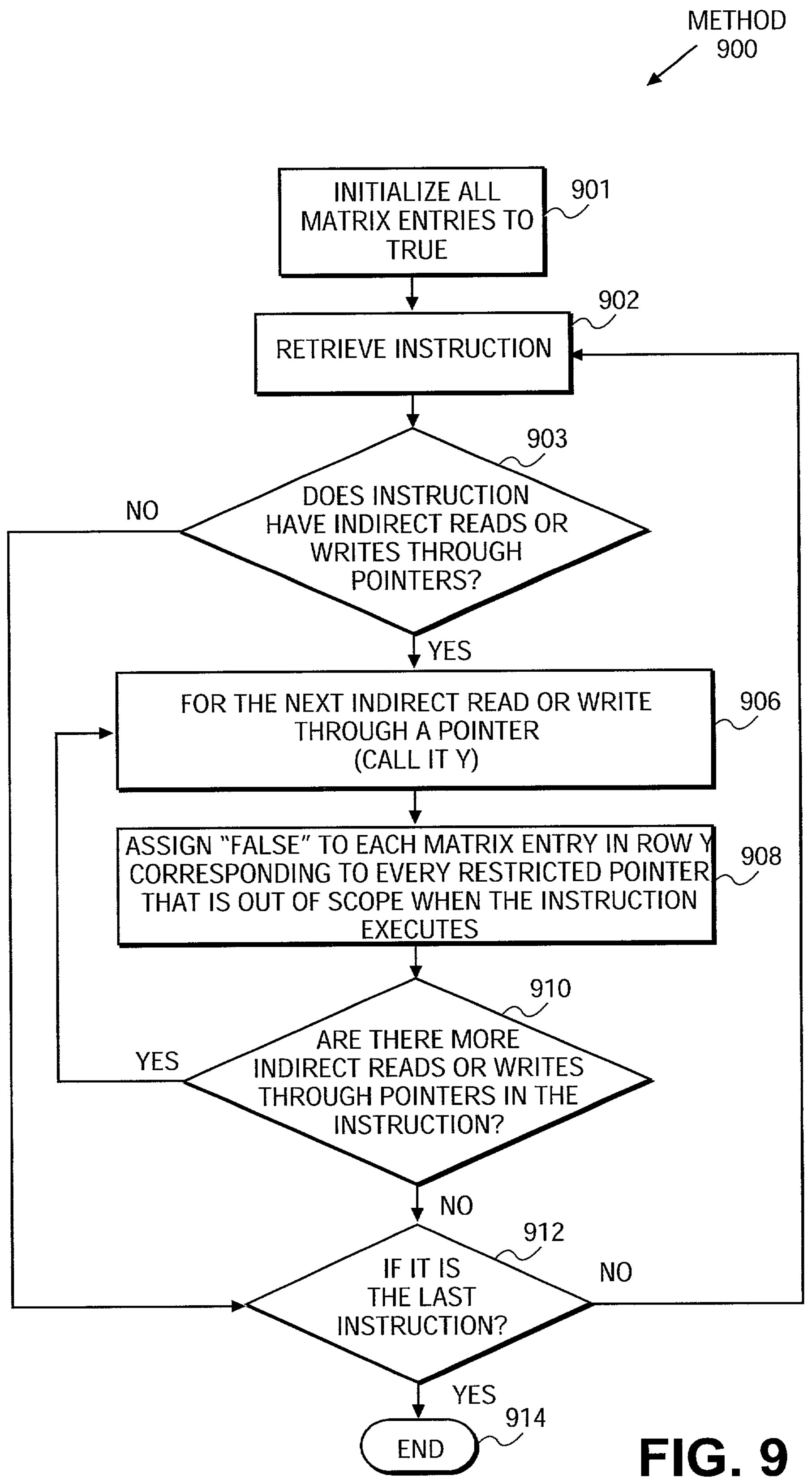
METHOD 900
INITIALIZE ALL MATRIX ENTRIES TO TRUE
901
RETRIEVE INSTRUCTION
902
DOES INSTRUCTION HAVE INDIRECT READS OR WRITES THROUGH POINTERS?
903
NO
YES
FOR THE NEXT INDIRECT READ OR WRITE THROUGH A POINTER (CALL IT Y)
906
ASSIGN "FALSE" TO EACH MATRIX ENTRY IN ROW Y CORRESPONDING TO EVERY RESTRICTED POINTER THAT IS OUT OF SCOPE WHEN THE INSTRUCTION EXECUTES
908
ARE THERE MORE INDIRECT READS OR WRITES THROUGH POINTERS IN THE INSTRUCTION?
910
YES
NO
IF IT IS THE LAST INSTRUCTION?
912
NO
YES
END
914

FIG. 9

```
procedure SCOPE_WALK

1010 for each i in D do
1015   for each j in R do
1020     MATRIX[ROW(i),j] := true;
1025   enddo
1030 enddo                              1002

1035 for each instruction x do
1040   for each indirect read or write through a pointer y do
1045     i := ROW(y);
1050     k : = REP(y).col;
1055     if k∈ (R∪P) then
1060       for each j in R do
1065         if j is not in scope when instruction x executes then
1070           MATRIX[i,j] := false;
1075         endif
1080       enddo
1090
1095     enddo
1096 endo                                                  1004

end SCOPE_WALK
```

Pseudo Code 1000

FIG. 10

	a	b	c	d	e
a		x	x	x	x
b				x	x
c				x	x
d		x	x		
e		x	x		
x		x	x	x	x
y				x	x

Matrix 1100

FIG. 11

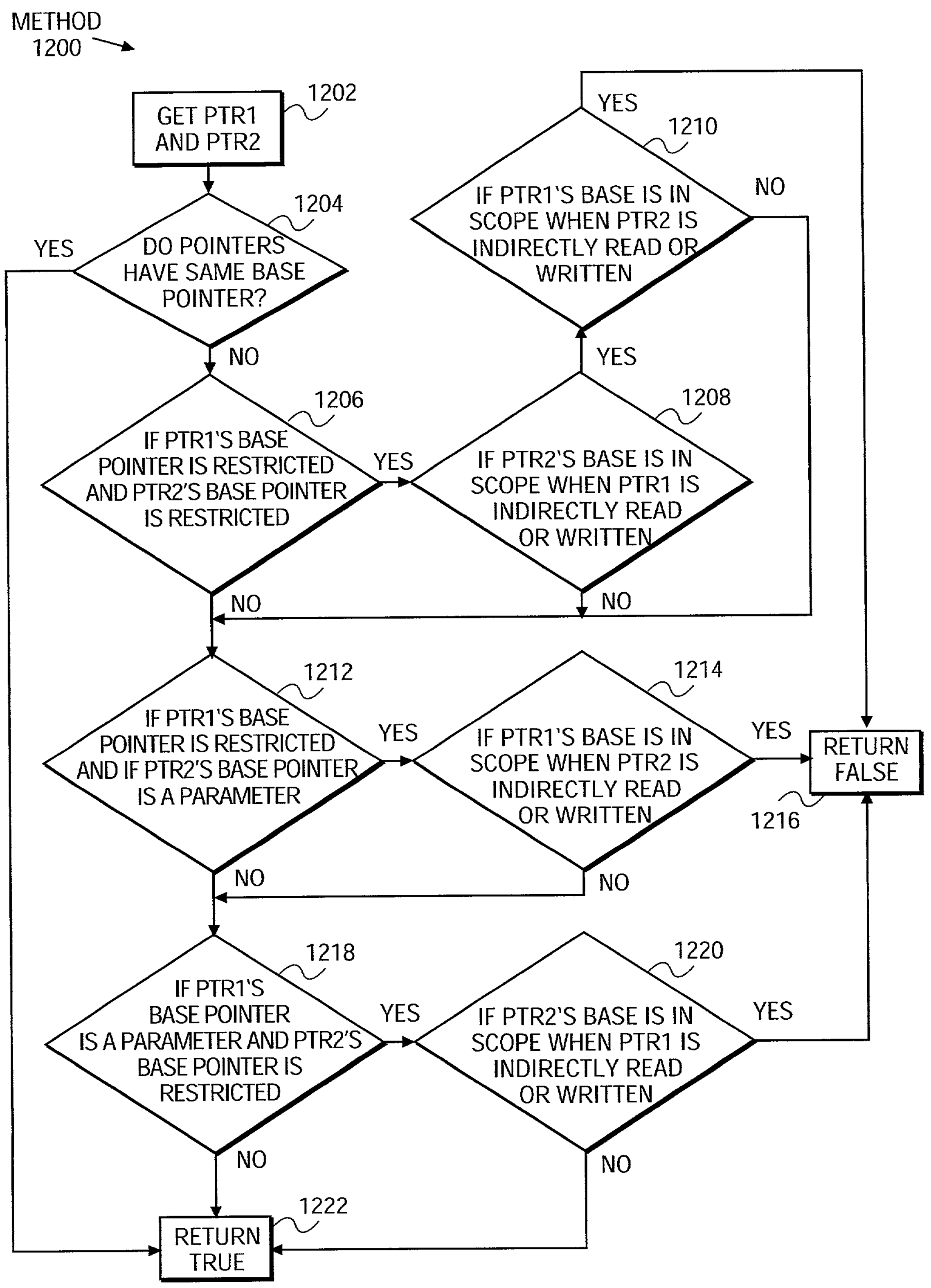
METHOD 1200
GET PTR1 AND PTR2
1202
DO POINTERS HAVE SAME BASE POINTER?
1204
YES
NO
IF PTR1'S BASE POINTER IS RESTRICTED AND PTR2'S BASE POINTER IS RESTRICTED
1206
YES
NO
IF PTR2'S BASE IS IN SCOPE WHEN PTR1 IS INDIRECTLY READ OR WRITTEN
1208
YES
NO
IF PTR1'S BASE IS IN SCOPE WHEN PTR2 IS INDIRECTLY READ OR WRITTEN
1210
YES
NO
IF PTR1'S BASE POINTER IS RESTRICTED AND IF PTR2'S BASE POINTER IS A PARAMETER
1212
YES
NO
IF PTR1'S BASE IS IN SCOPE WHEN PTR2 IS INDIRECTLY READ OR WRITTEN
1214
YES
NO
RETURN FALSE
1216
IF PTR1'S BASE POINTER IS A PARAMETER AND PTR2'S BASE POINTER IS RESTRICTED
1218
YES
NO
IF PTR2'S BASE IS IN SCOPE WHEN PTR1 IS INDIRECTLY READ OR WRITTEN
1220
YES
NO
RETURN TRUE
1222

FIG. 12

Pseudo Code 1300

```
procedure COULD_TARGETS_ALIAS(x,y)

    1320  i = REP(x).col;
    1302  j = REP(y).col;
    1306  if i=j then
    1308      return true;
    1310  endif                1302

    1312  if i∈R and j∈R and MATRIX[ROW(x),j]=true
    1314  and MATRIX[ROW(y),i]=true then
    1316      return false;
    1318 endif                                  1304

    1320  if i∈R and j∈P and MATRIX[ROW(y),i]=true then
    1322      return false;
    1324 endif
                                           1306

    1326  if j∈R and i∈P and MATRIX[ROW(x),j]=true then
    1328      returns false;
    1330  endif
    1340 return true;
                                           1308
    end COULD_TARGETS_ALIAS
```

FIG. 13